Brady R. Cornett

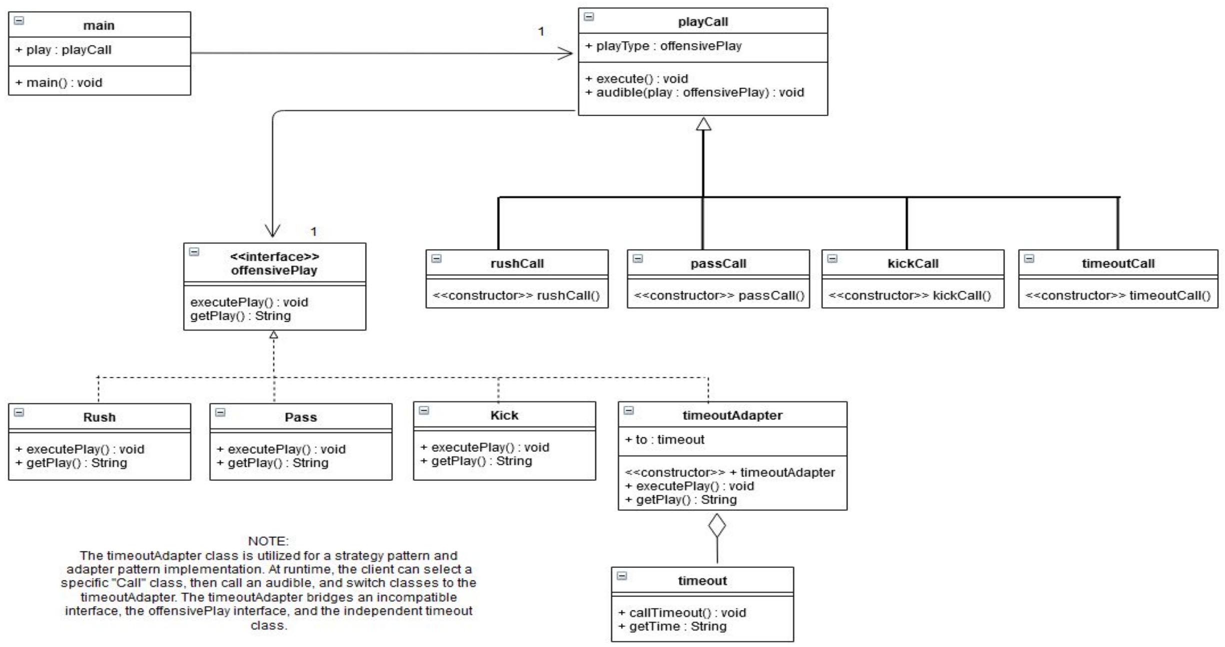
Software Engineering (ESOF-322)

Assignment 3

09/26/2019

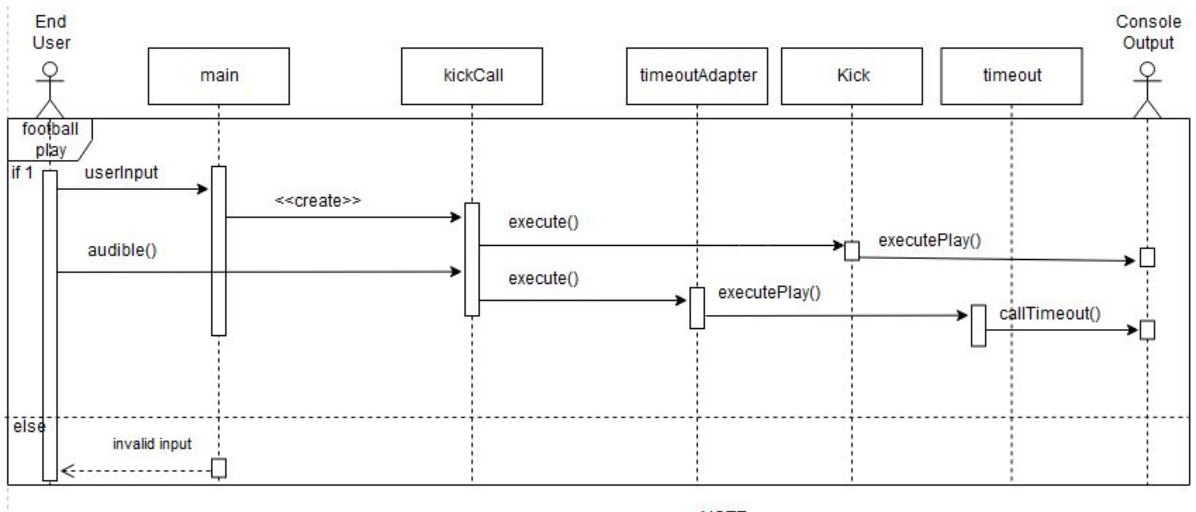
Exercise 1:

a)



NOTE:
The timeoutAdapter class is utilized for a strategy pattern and adapter pattern implementation. At runtime, the client can select a specific "Call" class, then call an audible, and switch classes to the timeoutAdapter. The timeoutAdapter bridges an incompatible interface, the offensivePlay interface, and the independent timeout class.

b)



NOTE:
Doing one sequence diagram where the user selects a kickCall, switches to the timeoutAdapter using audible function within kickCall. This demonstrates strategy and adapter pattern.

kickCall is a child class of parent class playCall. timeoutAdapter implements an interface

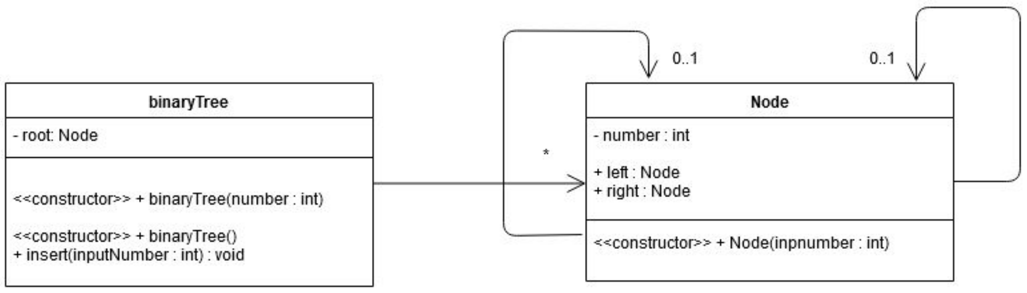
Exercise 2:

a) Factor: 32/45 = .71, Man-hours: 45 + 15 + 12 = 72. 72*.71 = *51.1 Story Points*

b) You can assume the focus factor of a brand new team to be around .7 or 70%.

c) Another way you could evaluate story points is asking each team member for their input and averaging the number. The problem with this is it takes away the open discussion that poker provides and veteran members may have different views on implementation which could skew the number.

d) and e)

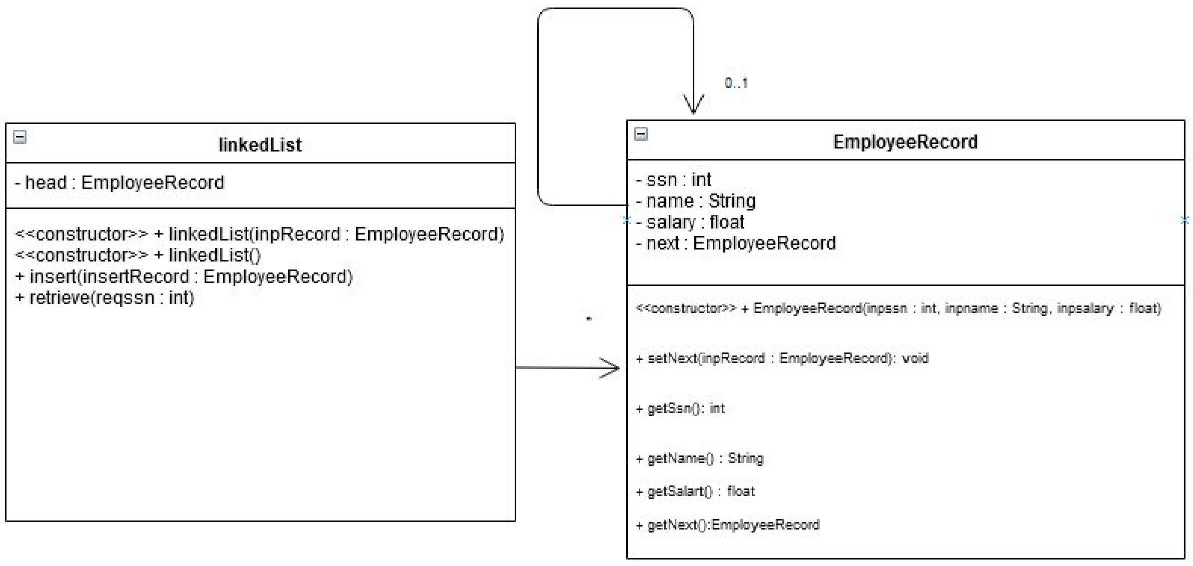


binaryTree
- root: Node
<<constructor>> + binaryTree(number : int)
<<constructor>> + binaryTree()
+ insert(inputNumber : int) : void
*
0..1
0..1
Node
- number : int
+ left : Node
+ right : Node
<<constructor>> + Node(inpnumber : int)

```
class binaryTree {
    private Node root;
    binaryTree(int number){
        root = new Node(number);
    }
    binaryTree(){
        root = null;
    }
    public void insert(int inputNumber){
        if (root == null){
            root = new Node(inputNumber);
        }else {
            Node iter = root;
            while (iter.left != null) {
                iter = iter.left;
            }
            iter.left = new Node(inputNumber);
        }
    }
}
```

```
class Node{
    private int number;
    Node left;
    Node right;
    public Node(int inpnumber){
        number = inpnumber;
        left = null;
        right = null;
    }
}
```

f) and g)


0..1
linkedList
- head : EmployeeRecord
<<constructor>> + linkedList(inpRecord : EmployeeRecord)
<<constructor>> + linkedList()
+ insert(insertRecord : EmployeeRecord)
+ retrieve(reqssn : int)
EmployeeRecord
- ssn : int
- name : String
- salary : float
- next : EmployeeRecord
<<constructor>> + EmployeeRecord(inpssn : int, inpname : String, inpsalary : float)
+ setNext(inpRecord : EmployeeRecord): void
+ getSsn(): int
+ getName() : String
+ getSalart() : float
+ getNext():EmployeeRecord

```
public class linkedList {
    private EmployeeRecord head;
    public linkedList(EmployeeRecord inpRecord){
        head = inpRecord;
    }
    public linkedList(){
        head = null;
    }
    public void insert(EmployeeRecord insertRecord){
        if (head == null) {
            head = insertRecord;
        }else{
            EmployeeRecord iter = head;
            while(iter.getNext() != null){
                iter = iter.getNext();
            }
            iter.setNext(insertRecord);
        }
    }
    public void retrieve(int reqssn){
        if (head == null){
            System.out.println("Empty");
        }else {
            EmployeeRecord iter = head;
            while(iter != null){
                if (iter.getSsn() == reqssn){
                    System.out.println(iter.getName());
                    break;
                }else{
                    iter = iter.getNext();
                }
            }
        }
    }
}

class EmployeeRecord{
    private int ssn; private String name; private float salary; private EmployeeRecord next;
    public EmployeeRecord(int inpssn, String inpname, float inpsalary){
        this.ssn = inpssn;
        this.name = inpname;
        this.salary = inpsalary;
    }
    public void setNext(EmployeeRecord inpRecord){
        this.next = inpRecord;
    }
    public int getSsn(){
        return this.ssn;
    }
    public String getName(){
        return this.name;
    }
    public float getSalary(){
        return this.salary;
    }
    public EmployeeRecord getNext(){
        return this.next;
    }
}
```